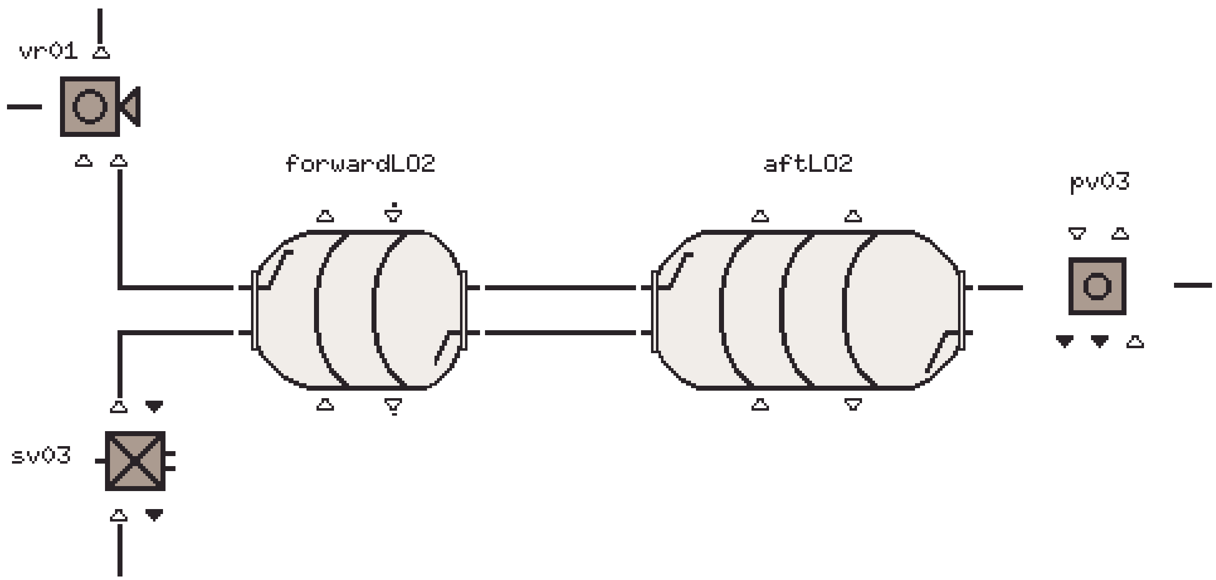
X-34 Demo Scope

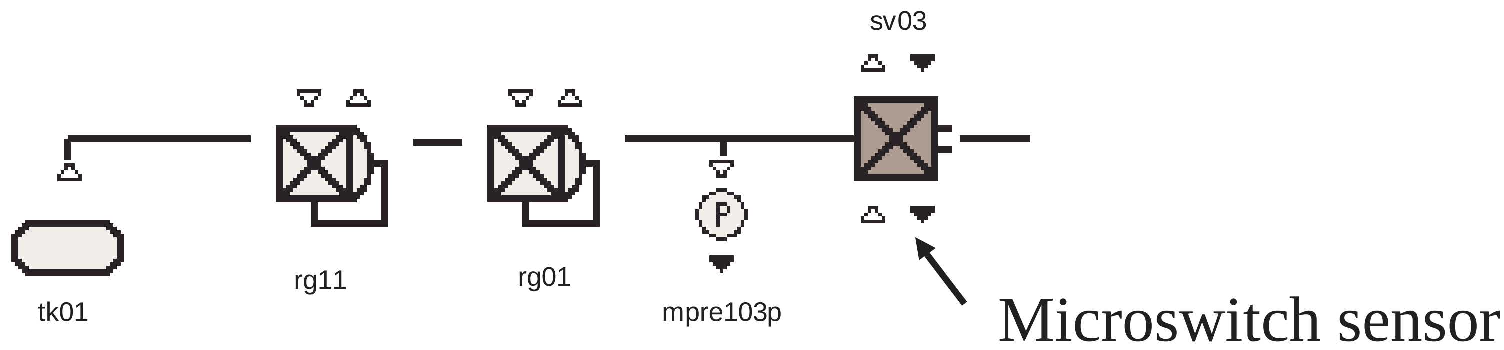
- X-34 model covers parts of the vehicle main propulsion system, during the captive carry phase of the flight
  - liquid oxygen (LOX) tank
  - the pressurization subsystem
  - the pneumatics subsystem
  - all sensors available in vehicle telemetry
- Does not include the kerosene propellant tank or the release, burn, and landing phases of the flight

The Liquid Oxygen tank



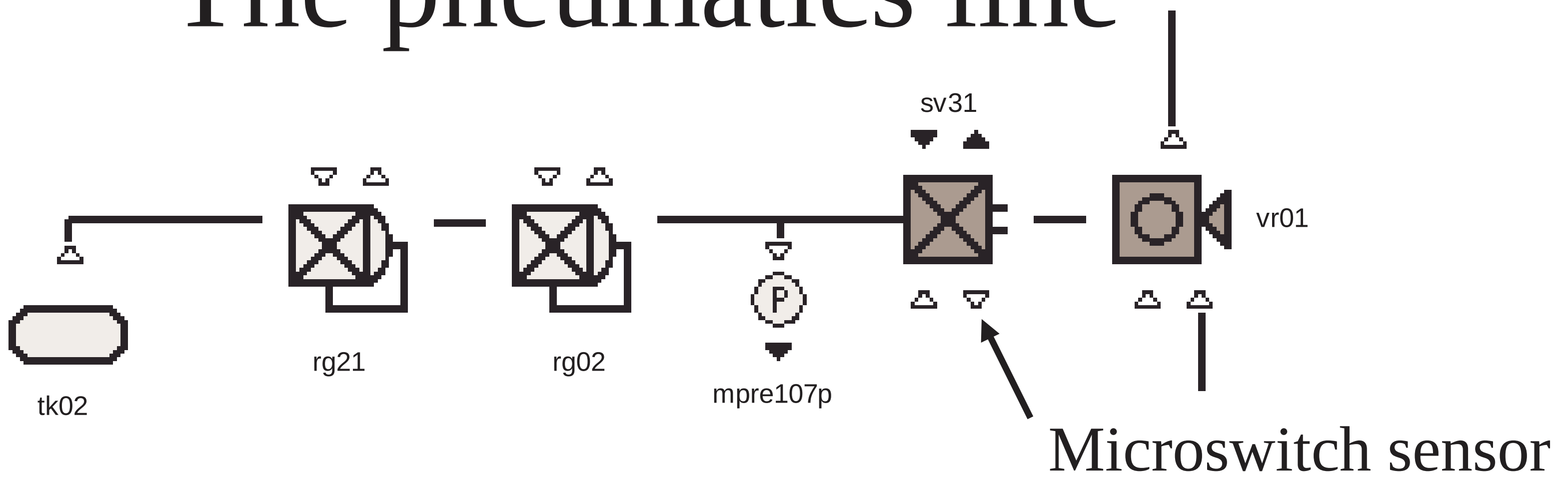
- Two components to the LOX tank, forward and aft
- Three valves to control pressurization, vent relief, and engine feed lines

The pressurization line



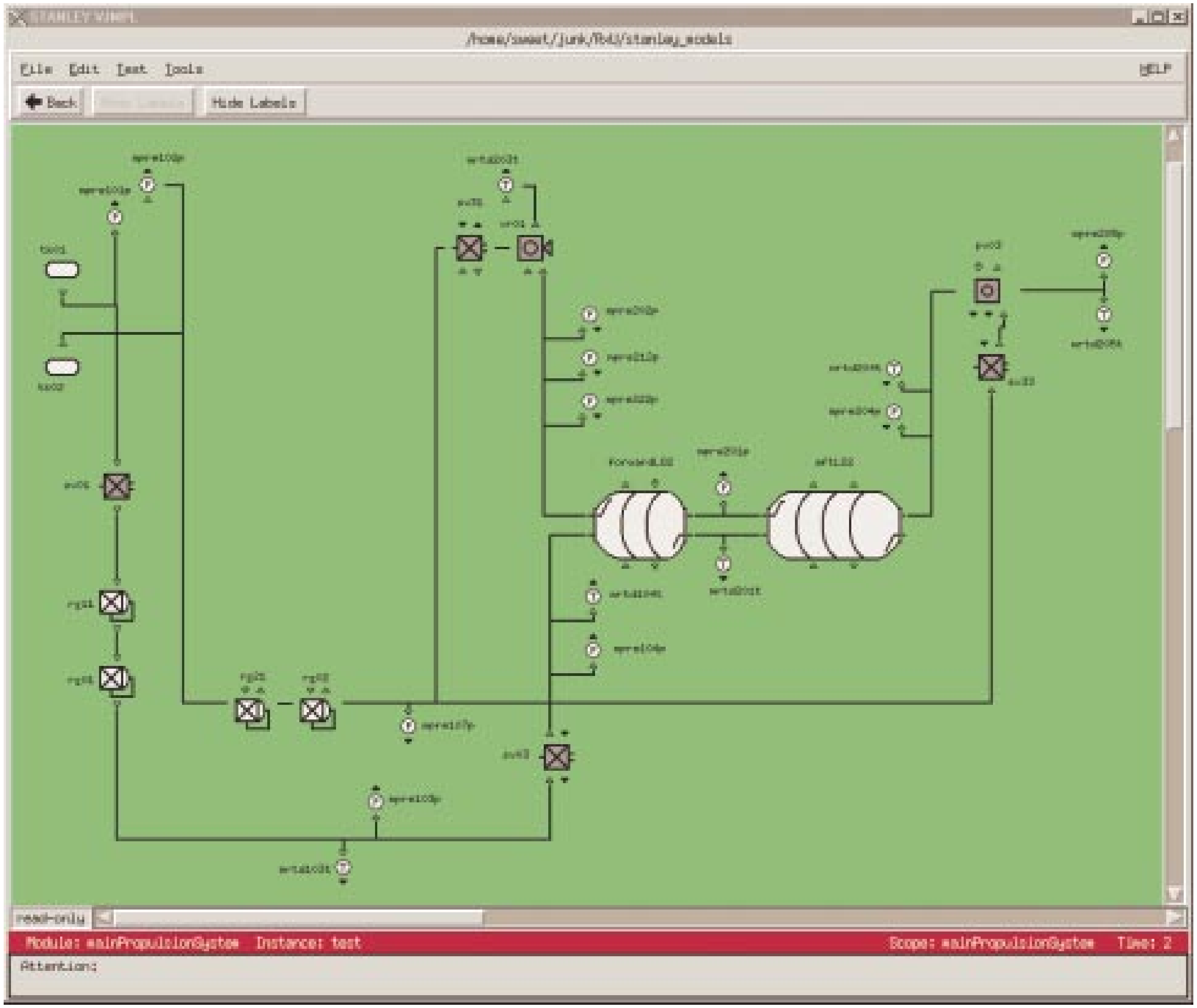
- Begins with tank containing gaseous He supply
- Two regulators on line provide redundancy and interesting diagnosis cases
- Solenoid pressurization valve regulates flow of He into LOX tank

The pneumatics line



- Similar to the pressurization line, begins with He tank and two regulators
- Solenoid valve actuates a large pneumatics valve
- Combination of regulators, solenoid valve, and pneumatic valve provide for interesting diagnosis

The full X-34 Livingstone model



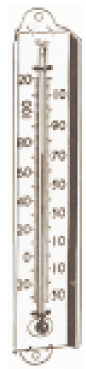
Data types for X-34 model

Threshold values	**Ranges**		**Datatypes**
aboveThreshold belowThreshold	upperBound	loxTemp	feedLineTemperature
aboveThreshold belowThreshold	lowerBound	ambientTemp	

feedLineTemperature.loxTemp.upperBound = belowThreshold

This system of threshold values and ranges is common to almost all datatypes in model

Monitors decide threshold values



T = 278.3 °R

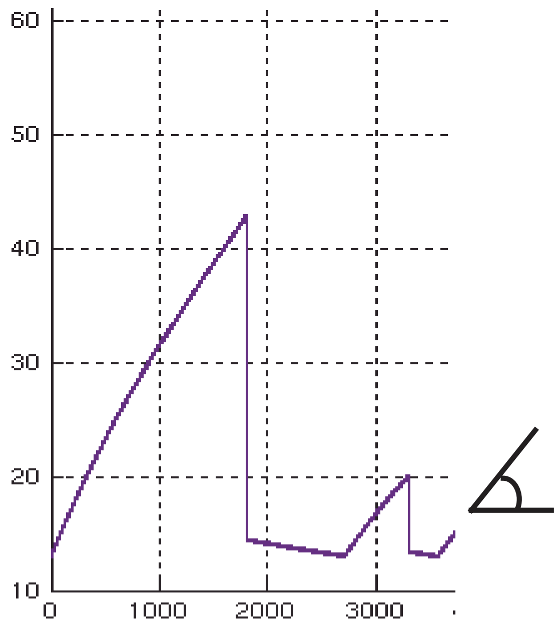
Monitor

LoxTemp upperBound threshold = 350 °R

feedLineTemperature.loxTemp.upperBound = belowThreshold

Thresholds on derivatives

- LOX tank state defined by the states of the three valves connecting to it
- Pressure derivative used to infer the state



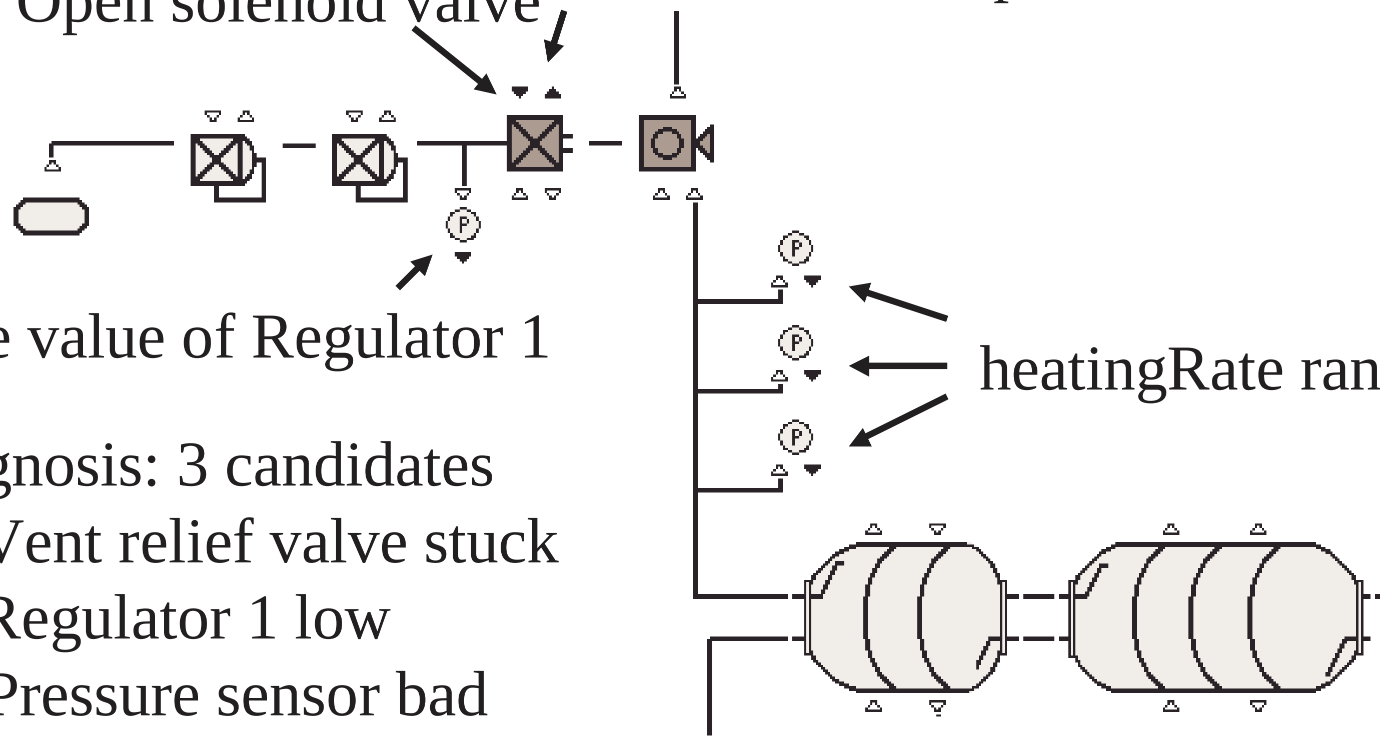
Derivative in pressurizationRate range $\Rightarrow$
pressurization valve open
Derivative in heatingRate range $\Rightarrow$
all valves closed
Derivative in bleedRate range $\Rightarrow$
engine outlet valve open
Derivative in ventingRate range $\Rightarrow$
vent relief valve open

Example of X-34 diagnosis

<u>Initial observations:</u>
<u>Final observations:</u>

<u>Command</u>: Open solenoid valve

Solenoid valve open
Solenoid valve closed



Pressure value of Regulator 1

heatingRate range

Diagnosis: 3 candidates
(1) Vent relief valve stuck
(2) Regulator 1 low
Pressure sensor bad
(3) Regulator 2 low
Pressure sensor bad
Probability of candidate determines order

X-34 Model summary

- Livingstone model contains a subset of the components of the X-34 main propulsion system
- Developed in the Stanley modeling environment
- Planned future work:
  - Expanding scope of model
- Possible future work
  - Language improvements
  - Unobservable commands

“Extra” slides I cut out to save space:

Overview

- What does Livingstone do?
- General Diagnosis example
- X-34 Scope and Description
- X-34 Diagnosis example
- Issues / Limitations

What does Livingstone do?

Livingstone is a diagnostic software tool. It tracks the state of a system based on the commands, the observations, and a model of the system.

- Livingstone is <u>not</u> a spacecraft controller, it advises a controller about the state of the system
- Created to be fast and memory-efficient by abstracting the system into a discrete domain
- Consistency-based reasoning allows Livingstone to do a diagnosis even if not all observations are fully known

What is a Livingstone model?

A Livingstone model describes a set of states for a system, the possible transitions between states, and the expected output based on the state.

- States are classified as “ok” and “failure”
- All quantities are discrete values

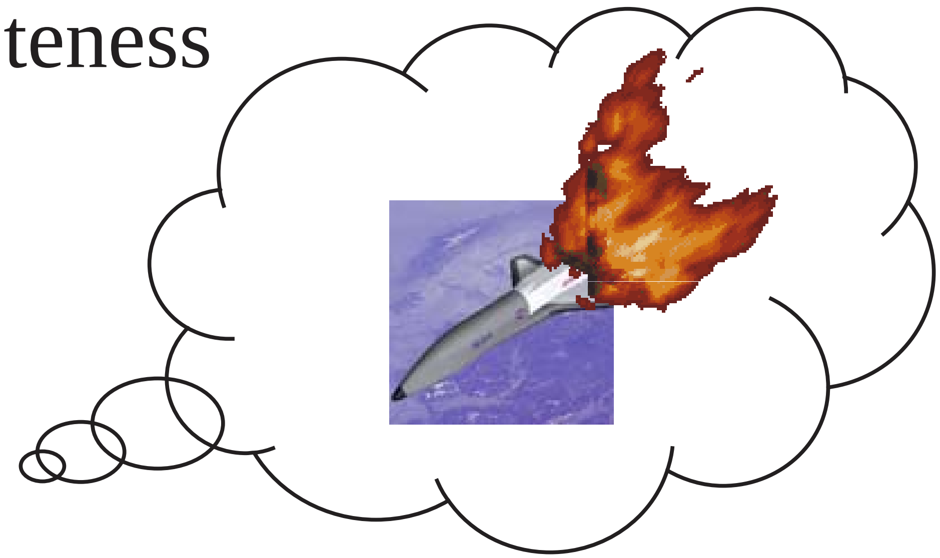
What is a Livingstone diagnosis?

A Livingstone diagnosis returns the set of states of a system that are consistent with the commands issued and the observed output.

- Takes into account the desired action via the commands
- Does not need all outputs of a system to do a diagnosis
- Probability of failure states is taken into account

Strategies for a “good” Livingstone model

- Component-based
- Predictive model
- Safety, not completeness



Example: House light model

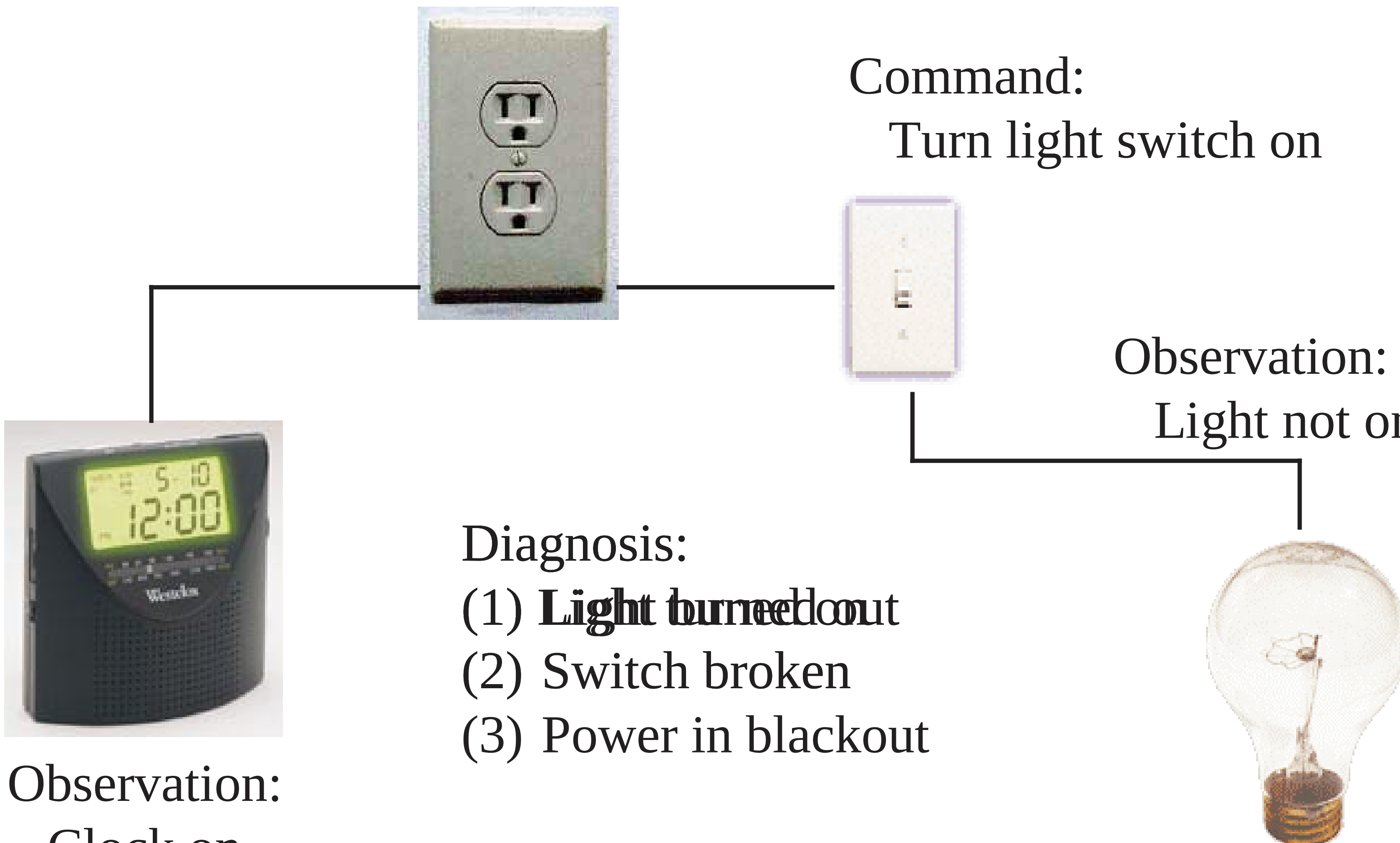
(1) Nominal
powerOut = on
(2) Rolling Blackout
<no constraints>

(1) On
powerOut = powerIn
(2) Off
powerOut = off
(3) Broken
<no constraints>
<u>Commands:</u> switch on/off

(1) Nominal
clock = powerIn
<u>Observation:</u> clock on/off

(1) Nominal
light = powerIn
(2) Burned-Out
<no constraints>
<u>Observation:</u> light on/off

House light example (2)



Command:
Turn light switch on

Observation:
Light not on

Observation:
Clock on

Diagnosis:
(1) Light burned out
(2) Switch broken
(3) Power in blackout

Model assumptions

- Modeled the main components of the system, but not all
- Some sensors do not have fault modes, will probably be added to next release

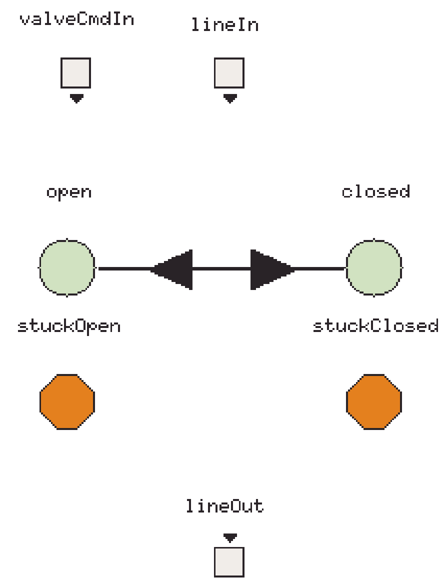
Failure of the Vent Relief Valve

Timing issues

- Often, a new command will be issued while we are still waiting for the results of the previous command to be issued. Have to deal with as a special case in the interface to Livingstone.
- Howard, are you going to talk about this?

Some issues

- Livingstone requires the modeler to abstract the system into a set of discrete states and the transitions between states. This works well for a valve model…



Some issues (3)

- Livingstone expects to have full knowledge of commands sent
- Commands for some components not in the telemetry stream visible to Livingstone
- Workaround in place, issue could drive future Livingstone research

What was I supposed to do again?

Some issues (2)

…but not as well for components with storage terms, like a tank of liquid.